

MENGUPAS RAHASIA  
DIBALIK SQL  
DDL-DML-DCL

# STRUCTURED QUERY LANGUAGE

— M FIKRI SETIADI —



## ***Kata Pengantar***

Puji syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa yang telah melimpahkan rahmat, karunia, serta hidayah-Nya sehingga penulis dapat menyelesaikan E-Book yang berjudul “**Kupas Tuntas SQL**” ini.

Sasaran dari E-Book ini adalah agar programmer, engineer, software developer, dan database programmer dapat memahami Syntax SQL dengan baik dan benar khususnya pemula.

Untuk mempermudah penulisan E-Book ini, penulis menggunakan Database Management System (DBMS) MySQL serta dibantu dengan tools SQLyog.

E-Book “**Kupas Tuntas SQL**” ini tidak hanya berlaku untuk DBMS MySQL, melainkan juga berlaku untuk semua DBMS seperti Oracle, SQL Server, Mongo DB, Maria DB, SQLite, dan lain-lainnya.

Akhirnya penulis berharap agar E-Book ini bermanfaat bagi pembacanya. Saran dan kritik sangat penulis harapkan untuk perbaikan E-Book ini.

Penulis

M Fikri Setiadi

# ***Daftar isi***

<b>Kata Pengantar .....</b>	<b>i</b>
<b>Daftar Isi.....</b>	<b>ii</b>
<b>Daftar Tabel .....</b>	<b>iv</b>
<b>Daftar Gambar.....</b>	<b>v</b>
 <b>Bab I. Structured Query Language (SQL)</b>	
1.1 Apa itu SQL .....	1
1.2 Sintaks .....	1
1.3 Nama .....	2
1.4 Type Data .....	2
1.5 Konstanta.....	6
1.6 Operator .....	6
1.7 Fungsi .....	7
 <b>Bab II. Data Definition Language (DDL)</b>	
2.1 CREATE .....	9
2.2 ALTER .....	11
2.3 DROP .....	13
 <b>Bab III. Data Manipulation Language (DML)</b>	
3.1 INSERT .....	14
3.2 UPDATE.....	17
3.3 SELECT .....	19
3.4 DELETE .....	29
 <b>Bab IV. Data Control Language (DCL)</b>	
4.1 GRANT.....	31
4.2 REVOKE .....	32
 <b>Bab V. SQL Tingkat Lanjut</b>	
5.1 COUNT .....	39

5.2 SUM.....	40
5.3 NOW.....	40
5.4 CURDATE.....	41
5.5 DAY.....	42
5.6 DATE .....	43
5.7 YEAR .....	44
5.8 JOIN.....	45
5.9 DATE_FORMAT .....	47
5.10 BETWEEN.....	48
5.11 GROUP BY .....	48
5.12 HAVING.....	49
5.13 VIEW .....	50
5.14 TRIGGER .....	52

## ***Daftar Tabel***

Tabel 1.1	Sintaks SQL.....	2
Tabel 1.2	Operator.....	7
Tabel 1.3	Fungsi.....	7

## ***Daftar Gambar***

Gambar 3.1	SELECT (*) .....	20
Gambar 3.2	SELECT CUSTOMIZE .....	20
Gambar 3.3	ORDER BY ASC .....	21
Gambar 3.4	ORDER BY DESC .....	21
Gambar 3.5	ORDER BY HARGA DESC .....	22
Gambar 3.6	WHERE BARANG_KODE .....	23
Gambar 3.7	WHERE BARANG_NAMA .....	23
Gambar 3.8	WHERE NOT BARANG_KODE .....	24
Gambar 3.9	WHERE BARANG_HARGA ANTARA 10000 S/D 20000 .....	24
Gambar 3.10	LIMIT RECORD .....	25
Gambar 3.11	LIMIT RECORD AFTER .....	26
Gambar 3.12	LIKE BERDASARKAN BARANG_NAMA .....	26
Gambar 3.13	LIKE DARI DEPAN KEYWORD .....	27
Gambar 3.14	LIKE DARI BELAKANG KEYWORD .....	28
Gambar 3.15	LIKE LEBIH DARI SATU FIELD .....	28
Gambar 5.1	TABLE KATEGORI .....	34
Gambar 5.2	TABLE BARANG .....	35
Gambar 5.3	TABLE JUAL .....	37
Gambar 5.4	TABLE DETAIL .....	38
Gambar 5.5	RELASI TABLE .....	39
Gambar 5.6	SELECT COUNT .....	40
Gambar 5.7	SELECT SUM .....	40
Gambar 5.8	SELECT NOW .....	41
Gambar 5.9	SELECT CURDATE .....	42
Gambar 5.10	SELECT DAY .....	42
Gambar 5.11	SELECT TODAY .....	43
Gambar 5.12	SELECT DATE .....	44
Gambar 5.13	SELECT YEAR .....	45
Gambar 5.14	SELECT JOIN DUA TABLE .....	46
Gambar 5.15	SELECT JOIN LEBIH DARI DUA TABLE .....	47
Gambar 5.16	SELECT DATE_FORMAT .....	47

Gambar 5.17	SELECT BETWEEN .....	48
Gambar 5.18	SELECT GROUP BY .....	49
Gambar 5.19	SELECT HAVING .....	50
Gambar 5.20	VIEW .....	51
Gambar 5.21	DATA TABLE BARANG.....	52
Gambar 5.22	TRIGGER .....	54



***1***

***Structured Query Language  
(SQL)***



## 1.1 Apa itu SQL

SQL adalah suatu bahasa computer yang mengikuti standar ANSI (*American National Standard Institute*), yaitu suatu bahasa standar yang digunakan untuk mengakses dan melakukan manipulasi sistem database. *Statement* dalam SQL dapat digunakan untuk mengakses data atau mengupdate data pada suatu database. Fungsi utama SQL digunakan untuk relational database seperti Oracle, SQL Server, DB2, Informix, Sybase, MS Access, MySQL, Mongo DB, SQLite, Maria DB, dan lain-lain.

Query adalah perintah SQL yang dirancang untuk memanggil kelompok record atau baris tertentu dari 1 tabel atau lebih dari suatu database. Bahasa Query mulai dikembangkan oleh IBM pada tahun 1970 yang hingga saat ini telah hampir digunakan oleh semua produk DBMS. Secara umum SQL dibagi menjadi 3 bagian yaitu: *Data Definition Language* (DDL), *Data Manipulation Language* (DML), dan *Data Control Language* (DCL). SQL selain merupakan query standar yang digunakan untuk mengakses relational database tetapi juga dapat mencakup kemampuan untuk mendefinisikan struktur data, perubahan data, pengaturan sekuritas dan lainnya. Elemen-elemen dalam SQL mencakup sintaks, nama, type data, konstanta, operator dan fungsi.

## 1.2 Sintaks

Sintaks adalah perintah SQL yang meminta suatu tindakan pada *Database Management System* (DBMS). SQL memiliki beberapa sintaks dasar sebagai berikut:

Tabel 1.1 Sintaks SQL

Sintaks	Keterangan
CREATE	Menciptakan table atau indeks
ALTER	Mengubah struktur table
DROP	Menghapus database atau table
INSERT	Menambahkan record dalam tabel
UPDATE	Mengubah record dalam tabel
SELECT	Memilih record dan field yang akan ditampilkan dari table
DELETE	Menghapus record dalam tabel

### 1.3 Nama

Nama ini digunakan sebagai identitas bagi objek-objek dalam DBMS, contohnya untuk objek-objek dalam DBMS tersebut adalah table, field, user.

### 1.4 Type data

Tipe data adalah skema pengkodean rinci yang dikendalikan oleh DBMS untuk mempresentasikan data yang terorganisasi. Setiap data memiliki tipe data, beberapa tipe data sebagai berikut:

#### 1) INT

INT merupakan type data angka (Integer) yang berukuran normal. Jangkauan nilainya adalah dari -2147483648 hingga 214783647.

2) BIGINT

Merupakan type data Integer (angka) yang berukuran besar. Jangkauan nilainya adalah dari -9223372036854775808 hingga 9223372036854775807.

3) SMALLINT

Merupakan type data Integer berukuran kecil. Jangkauannya dari -32768 hingga 32767.

4) MEDIUMINT

Merupakan type data integer tingkat menengah. Jangkauannya adalah dari -8388608 hingga 8388607.

5) TINYINT

Merupakan type data integer yang berukuran sangat kecil. Jangkauannya adalah dari -128 hingga 127.

6) FLOAT

Merupakan type data untuk bilangan floating-point presisi tunggal. Jangkauannya adalah dari -3.402823466E+38 hingga -1.175494351E-38 untuk bilangan negatif. Untuk bilangan positif antara 0, dan 1.175494351E-38 hingga 3.402823466E+38.

7) DOUBLE

Merupakan type data untuk bilangan berkoma presisi ganda. Jangkauannya adalah dari -1.7976931348623157E+308 hingga -2.2250738585072014E-308 untuk bilangan negatif. Untuk bilangan positif antara 0, dan 2.2250738585072014E-308 hingga 1.7976931348623157E+308.

8) DECIMAL

Bilangan decimal merupakan bilangan floating-point yang bersifat "unpacked". Yang berarti bilangan disimpan sebagai string, menggunakan satu karakter untuk setiap digitnya. Jangkauan DECIMAL sama dengan DOUBLE.

9) NUMERIC

Type data numeric sama dengan DECIMAL.

10) DATE

Merupakan type data tanggal (date). Jika di MySQL berformat 'YYYY-MM-DD'. Jangkauannya adalah antara nilai '1000-01-01' hingga '9999-12-31'.

11) DATETIME

Kombinasi dari waktu (time) dan tanggal (date). Jika di MySQL berformat 'YYYY-MM-DD HH:MM:SS'. Jangkauannya adalah dari '1000-01-01 00:00:00' hingga '9999-12-31 23:59:59'.

12) TIMESTAMP

Sebuah timestamp. Jangkauannya adalah dari '1970-01-01 00:00:00' hingga suatu waktu di tahun 2037.

13) TIME

Tipe data waktu, jangkauannya adalah '-838:59:59' hingga '838:59:59'. Time berformat 'HH:MM:SS'.

14) YEAR

Angka tahun, dalam format 2 digit atau 4 digit (default 4 digit). Nilai yang mungkin adalah 1901 hingga 2155. '0000' pada format 4 digit, dan 1970-2069 pada format 2 sigit (70-69).

15) CHAR

Merupakan type data string yang memiliki lebar tetap. Nilainya adalah dari 1 hingga 255 karakter. Jika ada sisa, maka sisa tersebut diisi dengan spasi. Dengan arti kata spasi dihitung sebagai karakter.

16) VARCHAR

Merupakan type data string yang memiliki lebar bervariasi. Nilainya adalah dari 1 hingga 255 karakter. Jika Anda mengset panjang karakternya 10 sedangkan data yang disimpan hanya terdiri dari 5 karakter, maka lebar data tersebut hanya 5 karakter saja. Dengan arti kata spasi tidak dihitung sebagai karakter.

17) TINYBLOB dan TINYTEXT

Sebuah BLOB (catatan atau gambar) atau TEXT dengan lebar maksimum 255 ( $2^8 - 1$ ) karakter.

18) BLOB dan TEXT

Sebuah BLOB atau TEXT dengan panjang karakter maksimum 65535 ( $2^{16}-1$ ) karakter. BLOB atau TEXT mendukung untuk karakter unik seperti derajat celcius ( $^{\circ}\text{C}$ ), Gender Male ( $\text{♀}$ ) atau Female ( $\text{♂}$ ), dan karakter unik lainnya.

19) MEDIUMBLOB dan MEDIUMTEXT

Sebuah BLOB atau TEXT dengan panjang karakter maksimum 16777215 ( $2^{24} - 1$ ) karakter.

20) LONGBLOB dan LONGTEXT

Sebuah BLOB atau TEXT dengan panjang karakter maksimum 4294967295 ( $2^{32} - 1$ ) karakter.

## 21) ENUM

Type data ini merupakan sebuah enumerasi, yaitu objek string yang hanya dapat memiliki nilai yang dipilih dari daftar nilai 'value1', 'value2', ....., NULL atau nilai special "" error. Sebuah ENUM maksimal dapat memiliki 65535 jenis nilai.

## 22) SET

Type data ini merupakan sebuah SET, yaitu objek string yang dapat memiliki 0 nilai atau lebih, yang harus dipilih dari daftar nilai 'value1', 'value2', ..... Sebuah SET maksimum dapat menampung 64 anggota.

### 1.5 Konstanta

Konstanta menyatakan nilai yang tetap (kembalikan konstanta adalah variabel). Beberapa contoh konstanta adalah sebagai berikut:

Konstanta Numerik : 123, -1345, 23.000

Konstanta : Jl. Bunda VI No 04

### 1.6 Operator

Operator adalah segala sesuatu yang digunakan untuk menghitung nilai yang akan menghasilkan nilai. Symbol-simbol yang dapat digunakan dalam operator yang dapat digunakan dalam operasi aritmatika dan perbandingan sebagai berikut:

Tabel 1.2 Operator

Simbol	Keterangan
*	Perkalian
/	Pembagian
+	Penambahan
-	Pengurangan
AND	Membandingkan 2 atau lebih variabel dengan setiap variable bernilai benar
OR	Membandingkan 2 atau lebih variabel dengan salah satu dari variable bernilai benar

## 1.7 Fungsi

Fungsi adalah sebuah sub program yang menghasilkan suatu nilai jika dipanggil. Adapun fungsi pada SQL adalah sebagai berikut:

Tabel 1.3 Fungsi

Fungsi	Keterangan
Min	Memperoleh nilai terkecil
Max	Memperoleh nilai terbesar
AVG	Memperoleh nilai rata-rata
SUM	Memperoleh total nilai
Count	Memperoleh total items
Date	Memperoleh nilai (yyyy-mm-dd)
Month	Memperoleh nilai bulan (mm)
Year	Memperoleh nilai tahun (yyyy)
Day	Memperoleh nilai hari (dd)
Data_add	Memperoleh nilai hasil operasi date sesuai dengan nilai interval
Date_format	Memperoleh nilai date yang terformat.

<b>datediff</b>	Memperoleh nilai hasil operasi date dengan operator aritmatika.
<b>Now</b>	Memperoleh nilai datetime sekarang.
<b>Curdate</b>	Memperoleh nilai date sekarang.
<b>IF</b>	Memperoleh nilai dari percabangan.



# 2

## *Data Definition Language (DDL)*



*Data definition language* atau biasa disingkat *DDL* adalah bagian dari *SQL (Structure Query Language)*. *DDL* berfungsi lebih kepada memanipulasi struktur database. *DDL* digunakan untuk membuat database, membuat tabel beserta struktur tabel. Mengubah struktur database, membuat relasi antar tabel, menghapus database, dan menghapus tabel. Berikut adalah perintah atau sintaks *Data Definition Language (SQL)*.

- *CREATE*, berfungsi untuk membuat database dan tabel.
- *ALTER*, berfungsi untuk mengubah struktur database.
- *DROP*, berfungsi untuk menghapus tabel dan database.

## **2.1 CREATE**

Sintak ini digunakan untuk membuat database dan membuat tabel beserta struktur tabel.

### **2.1.1 Membuat database (CREATE DATABASE)**

Untuk membuat database dengan perintah *CREATE DATABASE* bisa mengikuti pola sebagai berikut:

```
CREATE DATABASE nama_database;
```

Dimana *CREATE DATABASE* adalah sintak yang digunakan untuk membuat database, sedangkan *nama\_database* adalah nama dari database yang akan dibuat. Untuk penamaan database tidak boleh menggunakan spasi, untuk mengganti spasi gunakan underscore ( \_ ).

Contoh:

```
CREATE DATABASE db_penjualan;
```

Sintak diatas akan menghasilkan sebuah database dengan nama db\_penjualan.

### 2.1.2 Membuat tabel (CREATE TABLE)

Sintak CREATE juga bisa digunakan untuk membuat tabel dengan perintah CREATE TABLE dengan pola sebagai berikut:

```
CREATE TABLE tbl_barang(  
barang_kode VARCHAR(10) PRIMARY KEY,  
barang_nama VARCHAR(100),  
barang_satuan VARCHAR(20),  
barang_harga DOUBLE  
);
```

Sintak SQL diatas akan menghasil sebuah tabel bernama tbl\_barang dengan field (barang\_kode, barang\_nama, barang\_satuan, barang\_harga). Untuk varchar dan double adalah type data yang digunakan sedangkan angka yang ada dalam kurung adalah panjang karakter yang mampu ditampung oleh field tersebut dan primary key menandakan bahwa data yang ada pada field barang\_kode bersifat unik (tidak boleh sama).

Agar tabel yang kita buat mendukung untuk relational database. Maka harus menggunakan ENGINE InnoDB. Sedangkan sintak SQL diatas

menggunakan ENGINE default yaitu MyISAM. Untuk membuat tabel dengan ENGINE InnoDB adalah sebagai berikut:

```
CREATE TABLE tbl_barang(  
barang_kode VARCHAR(10) PRIMARY KEY,  
barang_nama VARCHAR(100),  
barang_satuan VARCHAR(20),  
barang_harga DOUBLE  
) ENGINE=INNODB;
```

## **2.2 ALTER**

Sintak ALTER berfungsi untuk mengubah struktur tabel. Sintak ini bisa digunakan untuk menambah field pada tabel, mengubah field pada tabel, dan menghapus field pada tabel.

### **2.2.1 Menambahkan field**

Sintak ALTER memungkinkan untuk menambahkan field yang tercecer pada suatu tabel. Adapun sintaknya adalah sebagai berikut:

```
ALTER TABLE tbl_barang ADD barang_berat INT;
```

Sintak diatas akan menghasilkan sebuah field barang\_berat yang disisipkan pada tabel tbl\_barang dengan type data INT (Integer). Secara default setiap field yang disisipkan menggunakan sintak ALTER akan diletakkan pada field terakhir. Sintak ALTER juga memungkinkan untuk menyisipkan field secara custom agar setiap field yang disisipkan tidak diletakkan paling akhir. Adapapun sintaknya adalah sebagai berikut:

```
ALTER TABLE tbl_barang ADD barang_jenis VARCHAR(20) AFTER  
barang_nama;
```

Sintak diatas akan menghasilkan sebuah field dengan nama barang\_jenis dengan type data varchar dan panjang karakternya (20) yang diletakkan setelah field barang\_nama.

### **2.2.2 Mengubah field**

Selain untuk menambahkan field pada tabel, sintak ALTER juga memungkinkan untuk mengubah field pada tabel. Adapun sintaknya adalah sebagai berikut:

```
ALTER TABLE tbl_barang CHANGE barang_jenis jenis_barang  
VARCHAR(20);
```

Sintak diatas akan mengubah field barang\_jenis menjadi jenis\_barang dengan type data varchar dan panjang karakternya (20).

### **2.2.3 Menghapus field**

Selain untuk menyisipkan dan mengubah field, sintak ALTER juga memungkinkan untuk menghapus field yang tidak diperlukan dan terlanjur dibuat. Adapaun sintaknya adalah sebagai berikut:

```
ALTER TABLE tbl_barang DROP barang_berat;
```

Sintak diatas akan menghapus field barang\_berat yang ada pada tabel tbl\_barang.

## **2.3 DROP**

Sintak DROP ini digunakan untuk menghapus tabel dan database.

Adapun sintaknya adalah sebagai berikut:

### **2.3.1 Menghapus database**

Adapun sintak untuk menghapus database adalah sebagai berikut:

```
DROP DATABASE db_penjualan;
```

Sintak diatas akan menghapus database dengan nama db\_penjualan.

### **2.3.2 Menghapus tabel**

Adapun sintak untuk menghapus tabel adalah sebagai berikut:

```
DROP TABLE tbl_barang;
```

Sintak diatas akan menghapus tabel dengan nama tbl\_barang.

# 3

## *Data Manipulation Language (DML)*



SQL adalah sintaks – sintaks atau perintah untuk mengakses data dalam database, tetapi SQL juga bisa digunakan untuk memanipulasi data dalam database seperti menambah data (*insert*), mengubah data (*update*), menampilkan data (*select*), dan menghapus data (*delete*). Sintaks – sintaks ini disebut *Data Manipulation Language* (DML) yang merupakan bagian dari SQL.

DML merupakan sekelompok perintah yang berfungsi untuk memanipulasi data dalam database, mulai dari menambah data, mengubah data, menampilkan data, dan menghapus data. Adapun penjelasannya sebagai berikut:

### **3.1 INSERT**

Sintaks INSERT merupakan perintah SQL yang berfungsi untuk menginputkan data kedalam tabel dari suatu database sesuai dengan struktur dari tabel tersebut.

Sebelum menginputkan data kedalam database, tentu saja harus ada database dan juga table. Kali ini penulis menggunakan database dan table pada bab sebelumnya. Adapun strukturnya sebagai berikut:

```
CREATE TABLE tbl_barang(  
barang_kode VARCHAR(10) PRIMARY KEY,  
barang_nama VARCHAR(100),  
barang_satuan VARCHAR(20),  
barang_harga DOUBLE  
);
```



Pada struktur tabel diatas terdapat 4 field yaitu **barang\_kode**, **barang\_nama**, **barang\_satuan**, dan **barang\_harga** dengan nama table yaitu **tbl\_barang**.

Sintaks INSERT memiliki dua macam pola dalam menambahkan data kedalam suatu tabel dalam database.

#### 1) **INSERT INTO VALUES ('value1','value2'.'....');**

Perintah SQL diatas akan menginputkan data kedalam suatu table dimana semua field yang ada pada tabel dan harus sesuai dengan urutan field dari suatu struktur tabel. Jika tidak akan menyebabkan data menjadi tidak valid.

Contoh:

```
INSERT INTO tbl_barang VALUES  
( 'BR001', 'Sampo', 'Unit', '12000' );
```

Perintah SQL diatas akan menambahkan data pada tabel **tbl\_barang** dimana **BR001** akan diinputkan ke field **barang\_kode**, **Sampo** diinputkan ke field **barang\_nama**, **Unit** diinputkan ke field **barang\_satuan**, dan **12000** diinputkan ke field **barang\_harga**. Data yang diinputkan diatas valid karena sesuai dengan urutan field yang ada pada table **tbl\_barang**. Sebaliknya jika Anda tidak memperhatikan strukturnya maka akan menyebabkan data tidak valid.

Contoh:

```
INSERT INTO tbl_barang VALUES  
( 'BR002', 'Unit', 'Sampo', '12000' );
```

Perintah SQL diatas akan menambahkan data pada tabel **tbl\_barang** dimana **BR002** akan diinputkan ke field **barang\_kode**, **Unit** diinputkan ke field **barang\_nama**, **Sampo** diinputkan ke field **barang\_satuan**, dan **12000** diinputkan ke field **barang\_harga**. Data yang diinputkan diatas tidak valid karena tidak sesuai dengan urutan field yang ada pada table **tbl\_barang**.

## 2) **INSERT INTO ('field1','field2','...') VALUES ('value1','value2'.'....');**

Perintah SQL diatas akan menginputkan data kedalam tabel dari suatu database, dimana tidak harus mengikuti urutan field dari struktur tabel ataupun harus menginputkan seluruh field yang ada pada tabel. Akan tetapi setiap value yang diinputkan harus mengikuti urutan field yang di SET.

Contoh:

```
INSERT INTO tbl_barang  
(barang_kode,barang_nama,barang_satuan,barang_harga)  
VALUES ('BR003','Sampo','Unit','12000');
```

Atau:

```
INSERT INTO tbl_barang  
(barang_kode,barang_satuan,barang_nama,barang_harga)  
VALUES ('BR003','Unit','Sampo','12000');
```

Pada sintaks pertama kita meletakkan field **barang\_nama** setelah **barang\_kode**, sedangkan pada sintaks kedua kita meletakkan field **barang\_nama** setelah **barang\_satuan**.

Perintah SQL diatas akan menambahkan data pada tabel **tbl\_barang** dimana **BR003** akan diinputkan ke field **barang\_kode**,

**Sampo** diinputkan ke field **barang\_nama**, **Unit** diinputkan ke field **barang\_satuan**, dan **12000** diinputkan ke field **barang\_harga**. Data yang diinputkan diatas **valid** karena **value** yang diinputkan sesuai dengan urutan field yang di **SET** bukan mengikuti urutan dari struktur pada table **tbl\_barang**.

Anda juga bisa menginputkan data tanpa harus menginputkan semua field yang ada pada suatu tabel dan data tetap valid.

Contoh:

```
INSERT INTO tbl_barang  
(barang_kode,barang_nama,barang_harga)  
VALUES ('BR004','Sampo','12000');
```

Perintah SQL diatas akan menambahkan data pada tabel **tbl\_barang** dimana **BR004** akan diinputkan ke field **barang\_kode**, **Sampo** diinputkan ke field **barang\_nama**, **12000** diinputkan ke field **barang\_harga**, dan menginputkan nilai kosong (NULL) pada field **barang\_satuan**.

### 3.2 UPDATE

Sintaks UPDATE merupakan perintah SQL yang berfungsi untuk mengubah / mengedit data dari satu tabel atau lebih dari suatu database.

Secara umum ada dua macam fungsi update pada SQL yaitu update secara keseluruhan dan update berdasarkan kondisi (**WHERE**). Adapun polanya sebagai berikut:

1) **UPDATE** nama\_table **SET** field1='value1', field2='value2', ...='...';

Pola update diatas akan mengupdate / mengubah data dari suatu tabel secara keseluruhan.

Contoh:

```
UPDATE tbl_barang SET  
barang_nama='Sabun', barang_harga='10000';
```

Perintah SQL diatas akan mengubah semua data yang ada pada table **tbl\_barang**. Dimana semua **barang\_nama** menjadi **Sabun** dan semua **barang\_harga** menjadi **10000**. Perintah SQL seperti ini memang sangat jarang digunakan dalam pemrograman. Akan tetapi penting untuk diketahui.

2) **UPDATE** nama\_table **SET** field1='value1', field2='value2', ...='...'

**WHERE** field\_kondisi='value\_kondisi';

Pola update diatas akan mengupdate / mengubah data dari suatu tabel berdasarkan kondisi tertentu.

Contoh:

```
UPDATE tbl_barang SET  
barang_nama='Garnier', barang_harga='18000' WHERE  
barang_kode='BR002';
```

Perintah SQL diatas hanya akan mengubah data yang ada pada table **tbl\_barang** yang memiliki **barang\_kode=BR002**. Dimana **barang\_nama** menjadi **Garnier** dan **barang\_harga** menjadi **18000**.

Secara teknis hanya mengubah satu data, karena field **barang\_kode** bersifat unik (**Primary Key**).

Selain itu Anda juga bisa mengubah data lebih dari satu dengan menggunakan operator pembandingan.

Contoh:

```
UPDATE tbl_barang SET  
barang_nama='Garnier2', barang_harga='19000' WHERE  
barang_kode='BR002' AND barang_kode='BR003';
```

Perintah SQL diatas akan mengubah dua data yaitu data dengan barang\_kode=BR002 dan BR003. Dimana **barang\_nama** menjadi **Garnier2** dan **barang\_harga** menjadi **19000**.

### 3.3 SELECT

SELECT merupakan perintah SQL yang berfungsi untuk menampilkan data kepada pengguna. Dengan menggunakan perintah SELECT ini memungkinkan Anda menyajikan informasi kepada *end-user* secara *customized* (sesuai kebutuhan).

Secara umum SELECT memiliki pola sebagai berikut:

```
SELECT nama_field FROM nama_tabel;
```

Penulis akan membagi select menjadi beberapa segmen sehingga membantu mempermudah Anda untuk memahaminya. Adapun pembagiannya sebagai berikut:

#### 1) **SELECT (\*)**

Select (\*) biasanya digunakan untuk menampilkan dari semua field yang ada pada suatu tabel. Tanda (\*) digunakan untuk memilih semua field yang ada pada suatu tabel.

Contoh:

```
SELECT * FROM tbl_barang;
```

Perintah SQL diatas akan menampilkan data dari semua field yang ada pada tabel **tbl\_barang**. Adapun data yang akan ditampilkan adalah sebagai berikut:

barang_kode	barang_nama	barang_satuan	barang_harga
BR001	Sabun	Unit	10000
BR002	Garnier2	Unit	19000
BR003	Garnier2	Unit	19000

Gambar 3.1. SELECT (\*)

Jika anda ingin menampilkan data secara customize. Anda cukup mengganti tanda bintang (\*) dengan nama field yang ingin Anda tampilkan datanya.

Contoh:

```
SELECT barang_nama, barang_harga FROM tbl_barang;
```

Perintah SQL diatas hanya akan menampilkan data **barang\_nama** dan **barang\_harga** dari tabel **tbl\_barang**. Adapun hasilnya seperti berikut:

barang_nama	barang_harga
Sabun	10000
Garnier2	19000
Garnier2	19000

Gambar 3.2. SELECT Customize

## 2) ORDER BY

ORDER BY berfungsi untuk mengurutkan (*sort*) data dari A-Z (ASC) atau dari Z-A (DESC). ASC singkatan dari *ascending* yaitu mengurutkan data dari yang terkecil hingga terbesar. Sedangkan DESC singkatan dari *descending* berfungsi untuk mengurutkan data dari yang terbesar sampai yang terkecil.

Contoh:

```
SELECT * FROM tbl_barang ORDER BY barang_kode ASC;
```

Perintah SQL diatas menampilkan data dari tabel **tbl\_barang** yang diurutkan berdasarkan **barang\_kode** dengan urutan dari yang terkecil hingga terbesar (ASC). Adapun hasilnya sebagai berikut:

barang_kode	barang_nama	barang_satuan	barang_harga
BR001	Sabun	Unit	10000
BR002	Garnier2	Unit	19000
BR003	Garnier2	Unit	19000

Gambar 3.3. ORDER BY ASC

Sebaliknya kita juga bisa mengurutkan data berdasarkan **barang\_kode** dari yang terbesar hingga terkecil (DESC).

Contoh:

```
SELECT * FROM tbl_barang ORDER BY barang_kode DESC;
```

Adapun hasilnya sebagai berikut:

barang_kode	barang_nama	barang_satuan	barang_harga
BR003	Garnier2	Unit	19000
BR002	Garnier2	Unit	19000
BR001	Sabun	Unit	10000

Gambar 3.4. ORDER BY DESC

Selain itu, anda juga bisa mengurutkan data berdasarkan **barang\_harga** tertinggi ataupun terendah.

Contoh:

```
SELECT * FROM tbl_barang ORDER BY barang_harga DESC;
```

Perintah SQL diatas akan menampilkan data dari harga tertinggi sampai terendah. Jadi Anda dapat melihat barang termahal dengan mudah. Adapaun hasilnya sebagai berikut:

barang_kode	barang_nama	barang_satuan	barang_harga
BR002	Garnier2	Unit	19000
BR003	Garnier2	Unit	19000
BR001	Sabun	Unit	10000

Gambar 3.5. ORDER BY HARGA DESC

Jika anda ingin mengurutkan data dengan harga termurah. Anda dapat mengganti DESC menjadi ASC pada perintah SQL diatas.

### 3) WHERE

WHERE berfungsi untuk menyeleksi data berdasarkan kondisi tertentu. Dengan menggunakan fungsi WHERE kita bisa menampilkan data yang diinginkan berdasarkan kondisi.

Contoh:

```
SELECT * FROM tbl_barang WHERE barang_kode='BR001';
```

Perintah SQL diatas akan menampilkan semua data dari setiap field yang ada pada tabel **tbl\_barang** berdasarkan kode barang BR001. Adapun hasilnya sebagai berikut:



barang_kode	barang_nama	barang_satuan	barang_harga
BR001	Sabun	Unit	10000

Gambar 3.6. WHERE BARANG\_KODE

Selain berdasarkan **barang\_kode**, Anda juga bisa menampilkan data berdasarkan **barang\_nama**.

Contoh:

```
SELECT * FROM tbl_barang WHERE barang_nama='Garnier2';
```

Perintah SQL diatas akan menampilkan semua data dari setiap field yang ada pada tabel **tbl\_barang** berdasarkan nama barang **Garnier2**. Adapun hasilnya sebagai berikut:

barang_kode	barang_nama	barang_satuan	barang_harga
BR002	Garnier2	Unit	19000
BR003	Garnier2	Unit	19000

Gambar 3.7. WHERE BARANG\_NAMA

Perintah SQL tersebut akan menampilkan dua data seperti gambar diatas. Artinya terdapat dua data dengan **barang\_nama** 'Garnier2' pada tabel **tbl\_barang**.

Selain kondisi diatas, anda juga bisa menampilkan data selain dari kondisi yang di tetapkan dengan menambahkan NOT pada kondisi.

Contoh:

```
SELECT * FROM tbl_barang WHERE NOT barang_kode='BR002';
```

Perintah SQL diatas akan menampilkan data selain dari **barang\_kode** 'BR002'. Adapun hasil sebagai berikut:

barang_kode	barang_nama	barang_satuan	barang_harga
BR001	Sabun	Unit	10000
BR003	Garnier2	Unit	19000

Gambar 3.8. WHERE NOT BARANG\_KODE

Anda juga bisa menampilkan data lebih dari satu kondisi. Dengan menambahkan operator logika ataupun operator perbandingan.

Contoh:

```
SELECT * FROM tbl_barang WHERE barang_harga > '10000'
AND barang_harga < '20000';
```

Perintah SQL diatas akan menampilkan semua data yang ada pada setiap field pada tabel **tbl\_barang** berdasarkan **barang\_harga** diatas **10000** dan dibawah **20000**. Akan tetapi tidak termasuk **10000** ataupun **20000**. Karena operator yang digunakan adalah besar dari (**>**) dan kecil dari (**<**). Adapun hasilnya sebagai berikut:

barang_kode	barang_nama	barang_satuan	barang_harga
BR002	Garnier2	Unit	19000
BR003	Garnier2	Unit	19000

Gambar 3.9. WHERE BARANG\_HARGA ANTARA 10000 s/d 20000

#### 4) LIMIT

LIMIT merupakan sintaks SQL yang berfungsi untuk membatasi jumlah baris (*record*) yang ingin ditampilkan. Mengingat jumlah data yang semakin banyak dan kompleks sehingga menyulitkan penelusuran data dan akan memperlambat proses penelusuran dikarenakan data yang di load sangat banyak dalam satu waktu. Dengan fungsi

LIMIT Anda dapat membatasi data yang akan di load sehingga akan mempercepat proses penelusuran data. Limit benar-benar bermanfaat dalam membuat suatu aplikasi berbasis database. Limit sering digunakan dalam membuat *pagination* dalam suatu halaman website, limit juga biasa digunakan dalam proses pengambilan keputusan seperti memilih tiga mahasiswa dengan nilai tertinggi, tiga produk terlaris, dan banyak lagi yang bisa dilakukan dengan fungsi LIMIT.

Contoh:

```
SELECT * FROM tbl_barang LIMIT 2;
```

Perintah SQL diatas akan menampilkan semua data dari setiap field yang ada pada tabel **tbl\_barang** dan membatasinya menjadi dua baris (*record*) dengan fungsi LIMIT. Adapun hasilnya sebagai berikut:

barang_kode	barang_nama	barang_satuan	barang_harga
BR001	Sabun	Unit	10000
BR002	Garnier2	Unit	19000

Gambar 3.10. LIMIT RECORD

Pada tabel **tbl\_barang** terdapat tiga record yaitu BR001, BR002, dan BR003. Sedangkan yang ditampilkan hanya BR001 dan BR002. Anda juga bisa membatasi record tanpa menampilkan dua record sebelumnya.

Contoh:

```
SELECT * FROM tbl_barang LIMIT 2,1;
```

Perintah diatas akan menampilkan satu record yaitu BR003 tanpa menampilkan dua record sebelumnya. Adapaun hasilnya sebagai berikut:

barang_kode	barang_nama	barang_satuan	barang_harga
BR003	Garnier2	Unit	19000

Gambar 3.11. LIMIT RECORD AFTER

Selain diatas, anda juga bisa mengkobinasikan dengan fungsi WHERE, ORDER BY, dan fungsi lainnya.

## 5) LIKE

LIKE merupakan sintaks SQL yang berfungsi untuk menampilkan data berdasarkan kemiripan / seperti. LIKE sering kali digunakan oleh programmer untuk membuat pencarian pada suatu aplikasi. LIKE tidak bisa berdiri sendiri, LIKE biasanya dibantu oleh fungsi WHERE.

Contoh:

```
SELECT * FROM tbl_barang WHERE barang_nama LIKE '%nier%';
```

Perintah SQL diatas akan menampilkan data dari tabel **tbl\_barang** dimana terdapat kemiripan kata 'nier' pada field **barang\_nama**. Adapun hasilnya sebagai berikut:

barang_kode	barang_nama	barang_satuan	barang_harga
BR002	Garnier2	Unit	19000
BR003	Garnier2	Unit	19000

Gambar 3.12. LIKE BERDASARKAN BARANG\_NAMA

Tanda persen ( % ) menunjukkan indikator kemiripan. Apakah kemiripan di indikasi dari awal karakter ataupun akhir karakter. Pada contoh diatas terdapat dua tanda persen ( % % ). Yaitu diawal dan diakhir *keyword* (kata kunci). Artinya indikator kemiripan diawal dan akhir karakter.

Jika anda ingin penelusuran lebih spesifik. Anda dapat mengindikator diawal ataupun diakhir keyword.

Contoh:

```
SELECT * FROM tbl_barang WHERE barang_nama LIKE 'sab%';
```

Perintah SQL diatas akan mencari kemiripan berdasarkan **barang\_nama** yang memiliki awalan 'sab'. Adapun hasilnya sebagai berikut:

barang_kode	barang_nama	barang_satuan	barang_harga
BR001	Sabun	Unit	10000

Gambar 3.13. LIKE DARI DEPAN KEYWORD

Selain itu Anda juga bisa mengindikasikan penelusuran berdasarkan akhiran.

Contoh:

```
SELECT * FROM tbl_barang WHERE barang_nama LIKE '%r2';
```

Perintah SQL diatas akan mencari kemiripan berdasarkan **nama\_barang** yang memiliki akhiran 'r2'. Adapun hasilnya sebagai berikut:

barang_kode	barang_nama	barang_satuan	barang_harga
BR002	Garnier2	Unit	19000
BR003	Garnier2	Unit	19000

Gambar 3.14. LIKE DARI BELAKANG KEYWORD

Selain yang telah penulis uraikan diatas. Anda juga bisa melakukan penelusuran berdasarkan kemiripan dari dua atau lebih field. Misalnya anda ingin melakukan pencarian data berdasarkan kode barang ataupun nama barang secara bersamaan. Anda cukup menambahkan operator pembandingan yaitu OR (atau).

Contoh:

```
SELECT * FROM tbl_barang WHERE barang_kode LIKE '%gar%'
OR barang_nama like '%gar%';
```

Perintah SQL diatas akan mencari kemiripan kata 'gar' pada **barang\_kode** atau **barang\_nama**. Dengan begitu anda bisa mencari barang berdasarkan kode barang ataupun nama\_barang.

Adapun hasilnya sebagai berikut:

barang_kode	barang_nama	barang_satuan	barang_harga
BR002	Garnier2	Unit	19000
BR003	Garnier2	Unit	19000

Gambar 3.15. LIKE LEBIH DARI 1 FIELD

Mungkin anda sulit membedakan bagaimana fungsi LIKE bekerja. Dengan data yang lebih banyak dan sering latihan Anda akan memahami bagaimana fungsi LIKE bekerja.

### 3.4 DELETE

DELETE merupakan perintah SQL yang berfungsi untuk menghapus record pada suatu tabel dari suatu database. Setiap aplikasi berbasis database tidak terluput dari fungsi delete.

Secara umum ada dua macam fungsi delete.

#### 1) **DELETE semua record.**

DELETE semua record memiliki pola sebagai berikut:

**DELETE FROM** nama\_tabel;

Delete semua record dari suatu tabel memang sangat jarang digunakan dalam membuat suatu aplikasi berbasis database. Akan tetapi penting untuk diketahui.

Contoh :

```
DELETE FROM tbl_barang;
```

Perintah SQL diatas akan menghapus semua data/record yang ada pada tabel **tbl\_barang**;

#### 2) **DELETE record berdasarkan kondisi.**

Untuk menghapus data / record berdasarkan kondisi memiliki pola sebagai berikut:

**DELETE FROM** nama\_tabel **WHERE** field\_kondisi='value\_kondisi';

Fungsi delete ini sangat dibutuhkan oleh programmer untuk menghapus data-data tertentu yang dianggap tidak diperlukan lagi.

Contoh:

```
DELETE FROM tbl_barang WHERE barang_kode='BR003';
```

Perintah SQL diatas akan menghapus data dari tabel **tbl\_barang** yang memiliki kode barang 'BR003'.



# 4

## *Data Control Language (DCL)*



Sebuah database biasanya memiliki banyak data dan juga multiple user dalam pengaksesan data. Sehingga menyebabkan keamanan database menjadi lemah disebabkan oleh faktor eksternal ataupun internal. Oleh sebab itu diperlukan pengaturan hak akses dan control data penuh terhadap suatu database sehingga dapat meminimalisir serangan dari luar (external) maupun dari dalam (Internal).

*Data Control Language* (DCL) merupakan sub perintah SQL yang digunakan untuk melakukan pengontrolan data dan server databasenya. Seperti memanipulasi user dan hak akses.

Adapun sintak SQL yang termasuk kedalam kategori DCL adalah sebagai berikut:

#### **4.1. GRANT**

GRANT merupakan perintah SQL yang digunakan untuk memberikan hak akses oleh administrator database kepada user (pengguna biasa). Hak akses tersebut bisa berupa CREATE (membuat), SELECT (menampilkan data), DELETE (menghapus), UPDATE (mengubah), dan hak khusus yang berkenaan dengan sistem databasenya.

Syntax GRANT memiliki pola sebagai berikut:

```
GRANT privileges ON nama_tabel TO nama_user;
```

Contoh:

```
GRANT select,insert,update,delete ON tbl_barang TO fikri;
```

Perintah SQL diatas akan memberikan hak akses kepada user fikri untuk menampilkan/mengakses data (SELECT), menambah (INSERT), mengubah (UPDATE), dan menghapus (DELETE). Dalam hal ini user fikri tidak dapat membuat tabel baru ataupun megubah struktur tabel (ALTER).

#### **4.2. REVOKE**

REVOKE merupakan perintah SQL yang berfungsi untuk menghilangkan atau mencabut hak akses yang telah diberikan kepada user tertentu oleh administrator database.

Syntax REVOKE memiliki pola sebagai berikut:

```
REVOKE privileges ON nama_tabel FROM nama_user;
```

Contoh:

```
REVOKE insert,update,delete ON tbl_barang FROM fikri;
```

Perintah REVOKE diatas menunjukkan bahwa sebagian hak akses dari user fikri dicabut kembali. Hak akses yang dicabut adalah hak akses menambah (INSERT), mengubah (UPDATE), dan menghapus (DELETE). Sementara user fikri masih bisa menampilkan data, karena hak akses SELECT tidak dicabut.

*5*

*SQL Tingkat Lanjut*



Pada bab sebelumnya penulis telah membahas tentang Data Definition Language (DDL), Data Manipulation Language (DML), dan Data Control Language (DCL). Pada bab ini penulis akan mengupas tuntas rahasia dibalik SQL pada MySQL seperti fungsi, kondisi, relasi, view, trigger, dan lainnya.

Untuk membantu mempermudah penulis dalam menjelaskan bab ini, penulis membuat sebuah database dan beberapa table didalamnya. Adapun database dan tabel yang penulis gunakan adalah sebagai berikut:

1. Database Inventori.

```
CREATE DATABASE db_inventori;  
USE db_inventori;
```

Perintah SQL diatas akan membuat sebuah database dengan nama db\_inventori dan mengaktifkannya dengan perintah USE db\_inventori.

2. Table Kategori.

```
CREATE TABLE tbl_kategori(  
    kategori_id INT(11) PRIMARY KEY,  
    kategori_nama VARCHAR(30)  
)ENGINE=INNODB;
```

Perintah SQL diatas akan membuat sebuah tabel dengan nama tbl\_kategori dengan field kategori\_id dan kategori\_nama dengan index (Primary Key) pada field kategori\_id.

Input beberapa data pada tabel kategori dengan mengeksekusi query berikut:

```
INSERT INTO tbl_kategori (kategori_id,kategori_nama)
VALUES ('1','Sampo'),('2','Makanan'),('3','Minuman');
```

Perintah SQL diatas merupakan perintah SQL multiple Insert dan akan menginputkan tiga data pada tabel **tbl\_kategori**.

Silahkan cek data yang diinputkan dengan perintah berikut:

```
SELECT * FROM tbl_kategori;
```

Akan terlihat hasilnya seperti gambar berikut:

kategori_id	kategori_nama
1	Sampo
2	Makanan
3	Minuman

Gambar 5.1. Table Kategori

### 3. Table Barang

```
CREATE TABLE tbl_barang(
barang_id VARCHAR(11) PRIMARY KEY,
barang_nama VARCHAR(30),
barang_satuan VARCHAR(30),
barang_harga double,
barang_kategori_id int(11),
FOREIGN KEY(barang_kategori_id)
REFERENCES tbl_kategori(kategori_id) ON UPDATE CASCADE
)ENGINE=INNODB;
```

Perintah SQL diatas akan membuat sebuah tabel dengan nama **tbl\_barang** dengan field **barang\_id**, **barang\_nama**, **barang\_satuan**, **barang\_harga**, dan **barang\_kategori\_id** yang berelasi (berhubungan) tabel **tbl\_kategori** dengan index (PRIMARY KEY) pada field **barang\_id** dan FOREIGN KEY pada field **barang\_kategori\_id**.

Input beberapa data pada tabel barang dengan mengeksekusi query berikut:

```
INSERT INTO tbl_barang (barang_id,barang_nama,barang_satuan,  
barang_harga,barang_kategori_id) VALUES  
( 'B001','Sampo Clear Men 100ml','Pcs',12000,1),  
( 'B002','Mie Goreng','Pcs',1900,2),  
( 'B003','Kratindaeng','Botol',5000,3);
```

Perintah SQL diatas merupakan perintah SQL multiple Insert dan akan menginputkan tiga data pada tabel **tbl\_barang**.

Cek data yang telah diinputkan dengan perintah SQL berikut:

```
SELECT * FROM tbl_barang;
```

Akan terlihat hasilnya seperti gambar berikut:

barang_id	barang_nama	barang_satuan	barang_harga	barang_kategori_id
B001	Sampo Clear Men 100ml	Pcs	12000	1
B002	Mie Goreng	Pcs	1900	2
B003	Kratindaeng	Botol	5000	3

Gambar 5.2. Table Barang

#### 4. Table Jual

```
CREATE TABLE tbl_jual(  
    jual_nofaktur VARCHAR(10) PRIMARY KEY,  
    jual_tanggal DATETIME  
)ENGINE=INNODB;
```

Perintah SQL diatas akan membuat sebuah tabel dengan nama **tbl\_jual** dengan field jual\_nofaktur dan jual\_tanggal dengan index (Primary Key) pada field jual\_nofaktur.

Input beberapa data pada tabel **tbl\_jual** dengan mengeksekusi query berikut:

```
INSERT INTO tbl_jual(jual_nofaktur,jual_tanggal)  
VALUES ('F001','2017-01-12'),('F002','2017-01-12'),  
('F003','2017-01-13'),('F004','2017-01-14'),  
('F005','2017-01-15'),('F006','2017-01-15');
```

Perintah SQL diatas akan menginputkan enam data pada tabel **tbl\_jual** dengan query multiple insert.

Hasilnya akan terlihat seperti gambar berikut:



jual_nofaktur	jual_tanggal
F001	2017-01-12
F002	2017-01-12
F003	2017-01-13
F004	2017-01-14
F005	2017-01-15
F006	2017-01-15

Gambar 5.3. Table Jual

#### 5. Table Detail Jual

```
CREATE TABLE tbl_detail(
  detail_id INT(11) PRIMARY KEY AUTO_INCREMENT,
  detail_barang_id VARCHAR(11),
  detail_disc DOUBLE,
  detail_qty INT(11),
  detail_jual_nofaktur VARCHAR(10),
  FOREIGN KEY(detail_barang_id) REFERENCES tbl_barang(barang_id) ON
  UPDATE CASCADE,
  FOREIGN KEY(detail_jual_nofaktur) REFERENCES
  tbl_jual(jual_nofaktur) ON UPDATE CASCADE
)ENGINE=INNODB;
```

Perintah SQL diatas akan membuat sebuah tabel dengan nama **tbl\_detail** dengan field **detail\_id**, **detail\_barang\_id**, **detail\_disc**, **detail\_qty**, dan **detail\_jual\_nofaktur** yang berelasi (berhubungan) tabel **tbl\_barang**

dengan index (PRIMARY KEY) pada field **barang\_id** dan FOREIGN KEY pada field **detail\_barang\_id**. Juga berelasi dengan tabel **tbl\_jual** dengan index (PRIMARY KEY) pada field **jual\_nofaktur** dan FOREIGN KEY pada field **detail\_jual\_nofaktur**.

Inputkan beberapa data pada **tbl\_detail** dengan mengeksekusi perintah SQL berikut:

```
INSERT INTO tbl_detail(detail_barang_id,detail_disc,detail_qty,
detail_jual_nofaktur) VALUES
('B001',1000,2,'F001'),('B002',0,1,'F001'),
('B001',1000,2,'F002'),('B003',0,1,'F002'),
('B001',1000,2,'F003'),('B002',0,1,'F003'),
('B002',1000,2,'F004'),('B003',0,1,'F004'),
('B001',1000,2,'F005'),('B002',0,1,'F005'),
('B003',1000,2,'F005'),('B002',0,1,'F006'),
('B003',1000,2,'F006'),('B002',0,1,'F006');
```

Perintah SQL diatas akan menginputkan 14 data kedalam **tbl\_detail**.

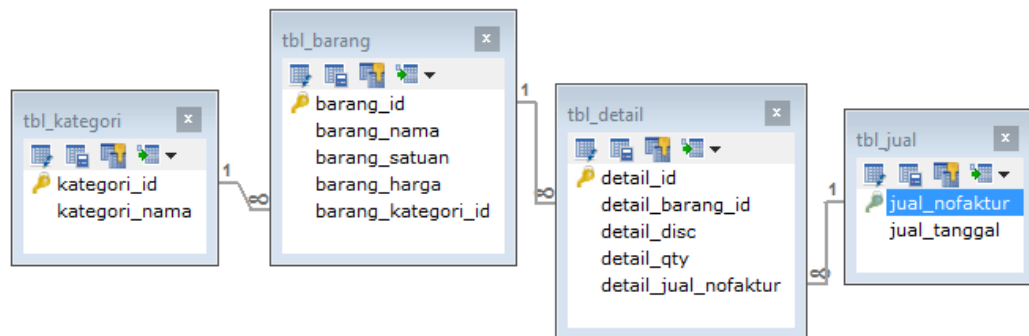
Adapaun hasilnya sebagai berikut:

detail_id	detail_barang_id	detail_disc	detail_qty	detail_jual_nofaktur
1	B001	1000	2	F001
2	B002	0	1	F001
3	B001	1000	2	F002
4	B003	0	1	F002
5	B001	1000	2	F003
6	B002	0	1	F003
7	B002	1000	2	F004
8	B003	0	1	F004
9	B001	1000	2	F005
10	B002	0	1	F005
11	B003	1000	2	F005
12	B002	0	1	F006
13	B003	1000	2	F006
14	B002	0	1	F006

Gambar 5.4. Table Detail

Sejauh ini proses pembuatan database, tabel, dan penginputan data bisa dinyatakan cukup.

Adapun bentuk relasi antar tabel dari database inventori yang telah dibuat adalah sebagai berikut:



Gambar 5.5. Relasi Table

## 5.1. COUNT

COUNT merupakan perintah SQL yang berfungsi untuk menghitung record (baris) pada suatu tabel atau lebih.

COUNT memiliki pola sebagai berikut:

```
SELECT COUNT(nama_field) AS nama_alias FROM nama_table;
```

Contoh penggunaan:

```
SELECT COUNT(*) AS tot_record FROM tbl_barang;
```

Perintah SQL diatas akan menampilkan total record dari table **tbl\_barang** dengan nama alias **tot\_record**.

Adapun hasil sebagai berikut:

tot_record
3

Gambar 5.6. Select COUNT

## 5.2. SUM

SUM merupakan Syntak SQL yang berfungsi untuk menjumlahkan field yang bernilai angka (INTEGER). SUM meiliki pola hampir sama dengan COUNT. Adapun contoh penggunaannya sebagai berikut:

```
SELECT SUM(barang_harga) AS tot_harga FROM tbl_barang;
```

Perintah SQL diatas akan menghitung total harga dari field **barang\_harga** dari tabel barang.

Adapun hasinya sebagai berikut:

tot_harga
18900

Gambar 5.7. Select SUM

## 5.3. NOW

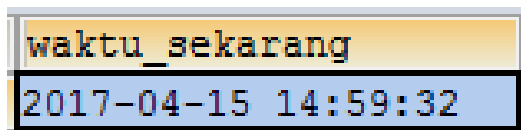
NOW merupakan waktu sekarang yang diambil dari server. NOW memiliki format **yyyy-MM-dd H:i:s**. Anda bisa menggunakan fungsi NOW untuk mengambil waktu sekarang jika Anda membutuhkannya.

Adapun contoh penggunaannya sebagai berikut:

```
SELECT NOW() AS waktu_sekarang;
```

Perintah SQL diatas akan menampilkan waktu sekarang degan nama alias waktu sekarang.

Adapun hasilnya sebagai berikut:



waktu_sekarang
2017-04-15 14:59:32

Gambar 5.8. Select NOW

#### 5.4. CURDATE

CURDATE merupakan perintah SQL untuk mengambil tanggal sekarang dari server. Berbeda dengan NOW, CURDATE memiliki format **yyyy-MM-dd**. CURDATE sangat bermanfaat dalam pembuatan suatu aplikasi, misalnya anda ingin menampilkan data tertentu berdasarkan tanggal sekarang. Anda dapat menggunakan CURDATE untuk melakukan hal itu.

Adapun contoh penggunaannya sebagai berikut:

```
SELECT CURDATE() AS tanggal_sekarang;
```

Adapun hasilnya sebagai berikut:

tanggal_sekarang
2017-04-15

Gambar 5.9. Select CURDATE

## 5.5. DAY

DAY merupakan perintah SQL yang berfungsi untuk mengambil hari pada field bertipe data DATE atau DATETIME.

Contoh:

```
SELECT DAY(jual_tanggal) AS hari FROM tbl_jual;
```

Perintah SQL diatas akan menampilkan hari dari field **jual\_tanggal** pada tabel **tbl\_jual** dengan nama alias **hari**.

Adapun hasilnya sebagai berikut:

hari
12
12
13
14
15
15

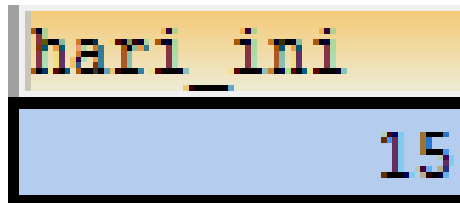
Gambar 5.10. Select DAY

Anda juga bisa menggunakan fungsi DAY untuk mendapat nilai hari ini dengan mengkombinasikan dengan fungsi CURDATE atau NOW.

Contoh:

```
SELECT DAY(CURDATE()) AS hari_ini;
```

Perintah SQL diatas akan menampilkan hari ini. Adapun hasilnya sebagai berikut:



Gambar 5.11. Select Today

## 5.6. DATE

DATE merupakan syntax SQL yang berfungsi untuk mengambil tanggal dari field bertipe data DATETIME.

Contoh:

```
SELECT DATE(jual_tanggal) AS tgl_jual FROM tbl_jual;
```

Perintah SQL diatas akan menampilkan tanggal jual dari tabel **tbl\_jual** dengan format **yyyy-MM-dd**. Fungsi DATE hanya akan menampilkan tanggal dari field **jual\_tanggal** yang bertipe DATETIME.

Adapun hasilnya sebagai berikut:

tgl_jual
2017-01-12
2017-01-12
2017-01-13
2017-01-14
2017-01-15
2017-01-15

Gambar 5.12. Select DATE

### 5.7. YEAR

YEAR merupakan perintah SQL yang berfungsi untuk menampilkan tahun pada field yang bertipe DATE atau DATETIME.

Contoh:

```
SELECT YEAR(jual_tanggal) AS thn_jual FROM tbl_jual;
```

Perintah SQL diatas akan menampilkan tahun pada field **jual\_tanggal** dengan nama alias **thn\_jual** data tabel **tbl\_jual**.



Adapun hasilnya sebagai berikut:

thn_jual
2017
2017
2017
2017
2017
2017

Gambar 5.13. Select YEAR

## 5.8. JOIN

JOIN merupakan syntax SQL yang berfungsi untuk menghubungkan beberapa table yang memiliki relasi atau hubungan.

Contoh:

```
SELECT barang_nama,barang_satuan,barang_harga,kategori_nama  
FROM tbl_barang JOIN tbl_kategori ON  
barang_kategori_id=kategori_id;
```

Perintah SQL diatas akan menghubungkan dua tabel yaitu antara tabel barang dengan tabel kategori dimana field **kategori\_id** sebagai PRIMARY KEY dan **barang\_kategori\_id** sebagai FOREIGN KEY.

Adapun hasil dari query diatas adalah sebagai berikut:

barang_nama	barang_satuan	barang_harga	kategori_nama
Sampo Clear Men 100ml	Pcs	12000	Sampo
Mie Goreng	Pcs	1900	Makanan
Kratindaeng	Botol	5000	Minuman

Gambar 5.14. Select JOIN dua tabel

Selain contoh diatas, JOIN juga bisa digunakan untuk menghubungkan dua tabel atau lebih.

Contoh:

```
SELECT jual_nofaktur,jual_tanggal,  
detail_barang_id,barang_nama,barang_satuan,  
barang_harga,kategori_nama,detail_disc,detail_qty  
FROM tbl_jual JOIN tbl_detail ON  
jual_nofaktur=detail_jual_nofaktur  
JOIN tbl_barang ON detail_barang_id=barang_id  
JOIN tbl_kategori ON barang_kategori_id=kategori_id;
```

Pada query diatas kita menghubungkan empat tabel sekaligus yaitu table barang, kategori, jual, dan detail. Dimana field jual\_nofaktur dan jual\_tanggal diambil dari tabel jual. Field detail\_barang\_id, detail\_disc, dan detail\_qty diambil dari tabel detail. Field barang\_nama, barang\_satuan, dan barang\_harga diambil dari table barang. Sedangkan field kategori\_nama diambil dari table kategori.

Jika query diatas di eksekusi, maka akan terlihat hasilnya sebagai berikut:

jual_nofaktur	jual_tanggal	detail_barang_id	barang_nama	barang_satuan	barang_harga	kategori_nama	detail_disc	detail_qty
F001	2017-01-12 00:00:00	B001	Sampo Clear Men 100ml	Pcs	12000	Sampo	1000	2
F002	2017-01-12 00:00:00	B001	Sampo Clear Men 100ml	Pcs	12000	Sampo	1000	2
F003	2017-01-13 00:00:00	B001	Sampo Clear Men 100ml	Pcs	12000	Sampo	1000	2
F005	2017-01-15 00:00:00	B001	Sampo Clear Men 100ml	Pcs	12000	Sampo	1000	2
F001	2017-01-12 00:00:00	B002	Mie Goreng	Pcs	1900	Makanan	0	1
F003	2017-01-13 00:00:00	B002	Mie Goreng	Pcs	1900	Makanan	0	1
F004	2017-01-14 00:00:00	B002	Mie Goreng	Pcs	1900	Makanan	1000	2
F005	2017-01-15 00:00:00	B002	Mie Goreng	Pcs	1900	Makanan	0	1
F006	2017-01-15 00:00:00	B002	Mie Goreng	Pcs	1900	Makanan	0	1
F006	2017-01-15 00:00:00	B002	Mie Goreng	Pcs	1900	Makanan	0	1
F002	2017-01-12 00:00:00	B003	Kratindaeng	Botol	5000	Minuman	0	1
F004	2017-01-14 00:00:00	B003	Kratindaeng	Botol	5000	Minuman	0	1
F005	2017-01-15 00:00:00	B003	Kratindaeng	Botol	5000	Minuman	1000	2
F006	2017-01-15 00:00:00	B003	Kratindaeng	Botol	5000	Minuman	1000	2

Gambar 5.15. Select JOIN lebih dari dua tabel

## 5.9. DATE\_FORMAT

DATE\_FORMAT merupakan syntax SQL yang berfungsi untuk memformat tanggal secara customize.

Contoh:

```
SELECT DATE_FORMAT(jual_tanggal,'%d %M %Y')
AS tanggal FROM tbl_jual;
```

Query diatas akan menghasil format tanggal seperti berikut:

tanggal
12 January 2017
12 January 2017
13 January 2017
14 January 2017
15 January 2017
15 January 2017

Gambar 5.16. Select DATE\_FORMAT

## 5.10. BETWEEN

BETWEEN merupakan syntax SQL yang berfungsi untuk menampilkan data dari periode ke periode tertentu.

Contoh:

```
SELECT jual_nofaktur,jual_tanggal,  
detail_barang_id,barang_nama,barang_satuan,  
barang_harga,detail_disc,detail_qty  
FROM tbl_jual JOIN tbl_detail ON  
jual_nofaktur=detail_jual_nofaktur  
JOIN tbl_barang ON detail_barang_id=barang_id  
WHERE jual_tanggal BETWEEN '2017-01-12' AND '2017-01-14';
```

Query diatas akan menampilkan data penjualan dari periode tanggal 12/01/2017 s/d 14/01/2017.

Adapun hasilnya sebagai berikut:

jual_nofaktur	jual_tanggal	detail_barang_id	barang_nama	barang_satuan	barang_harga	detail_disc	detail_qty
F001	2017-01-12 00:00:00	B001	Sampo Clear Men 100ml	Pcs	12000	1000	2
F002	2017-01-12 00:00:00	B001	Sampo Clear Men 100ml	Pcs	12000	1000	2
F003	2017-01-13 00:00:00	B001	Sampo Clear Men 100ml	Pcs	12000	1000	2
F001	2017-01-12 00:00:00	B002	Mie Goreng	Pcs	1900	0	1
F003	2017-01-13 00:00:00	B002	Mie Goreng	Pcs	1900	0	1
F004	2017-01-14 00:00:00	B002	Mie Goreng	Pcs	1900	1000	2
F002	2017-01-12 00:00:00	B003	Kratindaeng	Botol	5000	0	1
F004	2017-01-14 00:00:00	B003	Kratindaeng	Botol	5000	0	1

Gambar 5.17. Select BETWEEN

## 5.11. GROUP BY

GROUP BY merupakan syntax SQL yang berfungsi untuk pengelompokan data berdasarkan field tertentu.

Contoh:

```
SELECT jual_nofaktur,jual_tanggal,  
detail_barang_id,barang_nama,barang_satuan,  
barang_harga,SUM(detail_disc) AS disc,SUM(detail_qty) AS qty  
FROM tbl_jual JOIN tbl_detail ON  
jual_nofaktur=detail_jual_nofaktur  
JOIN tbl_barang ON detail_barang_id=barang_id GROUP BY  
jual_nofaktur;
```

Pada query diatas, penulis mengelompokan data berdasarkan no faktur dan menjumlahkan discount dan qty yang terdapat dalam satu faktur. Sehingga hasilnya terlihat seperti gambar berikut:

jual_nofaktur	jual_tanggal	detail_barang_id	barang_nama	barang_satuan	barang_harga	disc	qty
F001	2017-01-12 00:00:00	B001	Sampo Clear Men 100ml	Pcs	12000	1000	3
F002	2017-01-12 00:00:00	B001	Sampo Clear Men 100ml	Pcs	12000	1000	3
F003	2017-01-13 00:00:00	B001	Sampo Clear Men 100ml	Pcs	12000	1000	3
F004	2017-01-14 00:00:00	B002	Mie Goreng	Pcs	1900	1000	3
F005	2017-01-15 00:00:00	B001	Sampo Clear Men 100ml	Pcs	12000	2000	5
F006	2017-01-15 00:00:00	B002	Mie Goreng	Pcs	1900	1000	4

Gambar 5.18. Select GROUP BY

Agar mudah dipahami, silahkan eksekusi query diatas dengan menghilangkan **GROUP BY jual\_nofaktur**. Kemudian bandingkan hasil dengan menggunakan GROUP BY.

## 5.12. HAVING

HAVING merupakan syntax SQL yang berfungsi untuk menampilkan data yang memiliki kriteria tertentu.

Contoh:

```
SELECT jual_nofaktur,jual_tanggal,  
detail_barang_id,barang_nama,barang_satuan,  
barang_harga,detail_disc,detail_qty  
FROM tbl_jual JOIN tbl_detail ON  
jual_nofaktur=detail_jual_nofaktur  
JOIN tbl_barang ON detail_barang_id=barang_id  
HAVING detail_disc = 0;
```

Query diatas akan memfilter data penjualan yang tidak memiliki discount (discount=0).

Adapun hasilnya seperti gambar berikut:

jual_nofaktur	jual_tanggal	detail_barang_id	barang_nama	barang_satuan	barang_harga	detail_disc	detail_qty
F001	2017-01-12 00:00:00	B002	Mie Goreng	Pcs	1900	0	1
F003	2017-01-13 00:00:00	B002	Mie Goreng	Pcs	1900	0	1
F005	2017-01-15 00:00:00	B002	Mie Goreng	Pcs	1900	0	1
F006	2017-01-15 00:00:00	B002	Mie Goreng	Pcs	1900	0	1
F006	2017-01-15 00:00:00	B002	Mie Goreng	Pcs	1900	0	1
F002	2017-01-12 00:00:00	B003	Kratindaeng	Botol	5000	0	1
F004	2017-01-14 00:00:00	B003	Kratindaeng	Botol	5000	0	1

Gambar 5.19. Select HAVING

### 5.13. VIEW

VIEW merupakan perintah SQL yang berfungsi untuk menampilkan data dari query tertentu dengan nama view. View ini sangat bermanfaat untuk menampilkan data dari query yang kompleks agar mudah dalam pemanggilan. View juga bermanfaat untuk mempersingkat query untuk bahasa pemrograman yang membatasi jumlah karakter dalam satu baris kode seperti delphi.

View dapat dibuat dengan perintah CREATE VIEW.

Contoh:

```
CREATE VIEW view_jual_no_disc AS
SELECT jual_nofaktur,jual_tanggal,
detail_barang_id,barang_nama,barang_satuan,
barang_harga,detail_disc,detail_qty
FROM tbl_jual JOIN tbl_detail ON
jual_nofaktur=detail_jual_nofaktur
JOIN tbl_barang ON detail_barang_id=barang_id
HAVING detail_disc = 0;
```

Pada query diatas, penulis memuat sebuah view untuk query yang cukup kompleks dengan nama **view\_jual\_no\_disc**. Sehingga jika ingin menampilkan data penjualan yang tidak memiliki discount cukup dengan perintah berikut:

```
SELECT * FROM view_jual_no_disc;
```

Maka akan tampil data penjualan yang tidak memiliki discount, seperti gambar berikut:

jual_nofaktur	jual_tanggal	detail_barang_id	barang_nama	barang_satuan	barang_harga	detail_disc	detail_qty
F001	2017-01-12 00:00:00	B002	Mie Goreng	Pcs	1900	0	1
F003	2017-01-13 00:00:00	B002	Mie Goreng	Pcs	1900	0	1
F005	2017-01-15 00:00:00	B002	Mie Goreng	Pcs	1900	0	1
F006	2017-01-15 00:00:00	B002	Mie Goreng	Pcs	1900	0	1
F006	2017-01-15 00:00:00	B002	Mie Goreng	Pcs	1900	0	1
F002	2017-01-12 00:00:00	B003	Kratindaeng	Botol	5000	0	1
F004	2017-01-14 00:00:00	B003	Kratindaeng	Botol	5000	0	1

Gambar 5.20. VIEW

Jika ingin menghapus view, dapat dilakukan dengan perintah DROP VIEW.

Contoh:

```
DROP VIEW view_jual_no_disc;
```

#### 5.14. TRIGGER

TRIGGER atau pemicu merupakan perintah SQL yang berfungsi untuk menjalankan perintah DML (*Data Manipulation Language*) berdasarkan pemicu. Pemicunya dapat berupa AFTER (Setelah) atau BEFORE (Sebelum).

TRIGGER banyak digunakan oleh programmer untuk mengupdate stok barang ketika terjadi transaksi penjualan dalam aplikasi Point Of Sale.

Sebelum masuk kepembahasan lebih lanjut tentang trigger, silahkan ALTER dulu struktur dari tabel barang dan tambahkan satu field lagi yaitu **barang\_stok**. Kemudian inputkan nilai 10 pada field **barang\_stok**, sehingga terlihat data dari tabel barang yang baru menjadi seperti gambar berikut:

barang_id	barang_nama	barang_satuan	barang_harga	barang_kategori_id	barang_stok
B001	Sampo Clear Men 100ml	Pcs	12000	1	10
B002	Mie Goreng	Pcs	1900	2	10
B003	Kratindaeng	Botol	5000	3	10

Gambar 5.21. Data tabel barang

Nah, disini penulis akan membuat trigger sederhana dengan fungsi mengupdate stok pada tabel barang setelah tabel detail di input (INSERT).

Untuk membuat trigger dapat dilakukan dengan perintah CREATE TRIGGER.



Contoh:

```
DELIMITER $$

CREATE TRIGGER tr_jual AFTER INSERT ON tbl_detail
FOR EACH ROW BEGIN
    UPDATE tbl_barang SET
        barang_stok=barang_stok - NEW . detail_qty
    WHERE barang_id= NEW . detail_barang_id;
END$$

DELIMITER ;
```

Pada query diatas, penulis membuat sebuah trigger dengan nama **tr\_jual**, dimana trigger **tr\_jual** ini berfungsi untuk mengupdate stok pada tabel barang sesuai dengan qty yang ada pada tabel detail. Disini penulis menggunakan pemicu **AFTER INSERT** on **tbl\_detail**, artinya stok pada tabel barang akan diupdate disaat ada penginputan data pada tabel detail.

Untuk membuktikan apakah trigger berjalan dengan baik atau tidak, silahkan insert data pada tabel detail dengan mengeksekusi query berikut:

```
INSERT INTO tbl_detail
(detail_barang_id, detail_qty, detail_jual_nofaktur)
VALUES ('B001', '1', 'F006');
```

Setelah diinsert, cek data stok pada tabel barang dengan query berikut:

```
SELECT * FROM tbl_barang;
```

Maka akan terlihat hasilnya sebagai berikut:

barang_id	barang_nama	barang_satuan	barang_harga	barang_kategori_id	barang_stok
B001	Sampo Clear Men 100ml	Pcs	12000	1	9
B002	Mie Goreng	Pcs	1900	2	10
B003	Kratindaeng	Botol	5000	3	10

Gambar 5.22. Trigger

Pada gambar 5.21 terlihat bahwa stok barang dengan kode B001 telah berkurang setelah data pada tabel detail di input (INSERT).